路径操作

路径决定了文件或目录在文件系统中的位置，可以是绝对路径或相对路径。

绝对路径：从根目录开始的完整路径。比如：

> *D:\PythonFiles\p01.py*

相对路径：是相对于某个位置开始的路径。

> . 表示当前目录
> .. 表示当前目录的上一级目录

os模块中提供了很多对目录和文件操作的函数：

os.getcwd()

- 返回表示当前工作目录的字符串

```
import os


print(os.getcwd())
```

os.listdir(path)

- 返回 path 指定的文件夹中包含的文件或文件夹的名字构成的列表

```
import os


cwd = os.getcwd()
print(os.listdir(cwd))
```

os.makedirs(name, exist_ok=False)

- 创建目录，并且还会自动创建到达最后一级目录所需要的中间目录
- exist_ok 为 False (默认值)，表示如果目标目录已存在将引发 FileExistsError

```
import os


os.makedirs('./dir1/dir2/dir3')
```

os.path.basename(path)

- 返回路径 path 最后一级的名称，通常用来返回文件名

```
import os


print(os.path.basename('./dir3/dir2/dir1/a.txt'))
```

os.path.dirname(path)

- 返回路径 path 的目录名称

```
import os


print(os.path.dirname('./dir3/dir2/dir1/a.txt'))
```

os.path.split(path)

- 把路径分割成 dirname 和 basename，返回一个元组

```
import os


print(os.path.split('./dir3/dir2/dir1/a.txt'))
```

os.path.splitext(path)

- 把路径中的扩展名分割出来，返回一个元组

```
import os


print(os.path.splitext('./dir3/dir2/dir1/a.txt'))
```

os.path.exists(path)

- path 路径存在则返回 True，不存在则返回 False

```
import os


p = r'D:\PythonFiles\p01.py'
print(os.path.exists(p))
```

os.path.isfile(path)

- 判断路径是否为文件

```
import os


print(os.path.isfile("./dir3/dir2/dir1/a.txt"))
```

os.path.isdir(path)

- 判断路径是否为目录

```
import os


print(os.path.isdir("./dir3/dir2/dir1"))
```

os.path.join(path, *paths)

- 智能地拼接一个或多个路径部分

```python
import os



p1 = 'D:\\PythonFiles\\'
p2 = r'dir1\dir2\dir3'
p3 = 'p01.py'
print(os.path.join(p1, p2, p3))
```

文件读写

从文件的编码方式来看，文件可以分为文本文件和二进制文件。

文本文件：txt、html、json等； 二进制文件：图片、音频、视频等。

open(file, mode='r', encoding=None)

- file：文件路径
- mode：文件打开的模式，默认为 'r' 模式
- encoding：指定文本文件的编码方式，默认依赖系统，处理非ASCII文本时，"UTF-8" 通常是首选编码
- 打开指定的文件，返回一个文件对象（迭代器对象）

```
"""
当前open以默认模式打开指定的文件，返回一个文件对象，
该对象为迭代器对象，每次迭代会返回该文件中的一行数据
"""
file = open('./exam.txt')
print(next(file))

for i in file:
    print(i)
```

mode 常用模式：

模式	描述
r	以只读方式打开文件。文件的指针将会放在文件的开头。
w	打开一个文件只用于写入。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新的文件再写入。
a	打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。
+	如果要以读写模式打开，加上 + 即可，比如：r+、w+、a+

file 常用对象方法：

file.read(size=-1)

- 从 file 中读取至多 size 个字符并返回
- 如果 size 为负值或 None，则读取至 EOF（End Of File）

```
with open(r"./t01.txt") as file:
    print(file.read(5))
    print(file.read(2))
    print(file.read())
```

file.write(s)

- 将字符串 s 写入并返回写入的字符数

```
with open(r"./t01.txt", mode='a') as file:
    num = file.write('\nhello baby')
    print(num)
```

file.flush()

- 刷新缓冲区，即将缓冲区中的数据立刻写入文件，同时清空缓冲区，不需要被动的等待缓冲区写入。一般情况下，文件关闭后会自动刷新缓冲区，但有时你需要在关闭前刷新它，这时就可以使用 flush() 方法

```
import time



file = open(r"./t01.txt", mode='a')
file.write('\n123456789')
time.sleep(5)  # 文件需要等到关闭文件时才会把数据从缓冲区写入文件
file.close()  # 关闭文件，自动刷新缓冲区，数据才写入文件

file = open(r"./t01.txt", mode='a')
file.write('\n123456789')
file.flush()  # 刷新缓冲区，数据立刻写入文件
time.sleep(5)
file.close()
```

file.close()

- 刷新缓冲区并关闭该文件。如果文件已经关闭，则此方法无效
- 文件关闭后，对文件的任何操作（如：读取或写入）都会引发ValueError

```
file = open(r'./t01.txt')
print(file.read())
file.close()
file.read()  # 引发ValueError
```

file.seek(offset)

- 移动文件指针到指定位置

```
with open('./exam.txt', mode='w+') as file:
    file.write('hello\nworld')
    file.seek(7)
    print(file.read())  # 'world'
```

with 语句

- 这种写法如果在open之后close之前发生未知的异常，就不能确保打开的文件一定被正常关闭，这显然不是一个好的做法

```
file = open(r'./t01.txt', mode='w')
file.write('hello world')
...
...
file.close()
```

- 所以可以使用下面这种写法，确保close一定会被执行

```
file = open(r'./t01.txt', mode='w')
try:
    file.write('hello world')
    ...
    ...
finally:
    file.close()
```

- 用 with 语句将会是一种更加简洁、优雅的方式

```
with open(r'./t01.txt', mode='w') as file:
    file.write('hello world')
    ...
    ...
```